Processo – parte V

Leggere Sez. 7.7 Ghezzi et al, 17.1 e 17.2 Sommerville

università di ferrara
DA SEICENTO ANNI GUARDIAMO AVANTI.

Organizzazione del processo

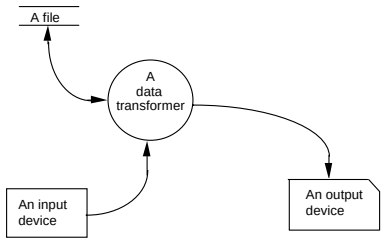
- Metodologie che definiscono
  - metodi
  - strumenti

  che supportano l'intero processo software.
- Linee guida per
  - singole fasi
  - transizioni fra fasi

Structured Analysis / Structured Design (SA/SD)

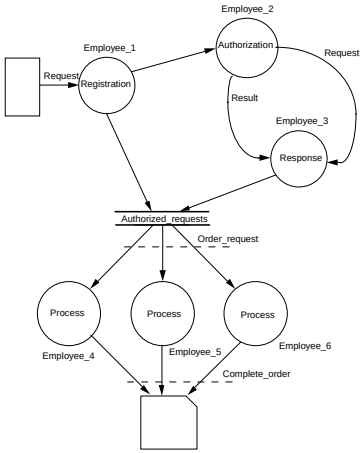
- Per fasi di analisi e progettazione
- Strumenti per analisi:
  - Data Flow Diagrams
  - Data Dictionaries: elenco di nomi e descrizioni per le entità nel sistema
  - Structured English: sottoinsieme del linguaggio naturale per descrivere le trasformazioni corrispondenti alle funzioni elementari

DFD

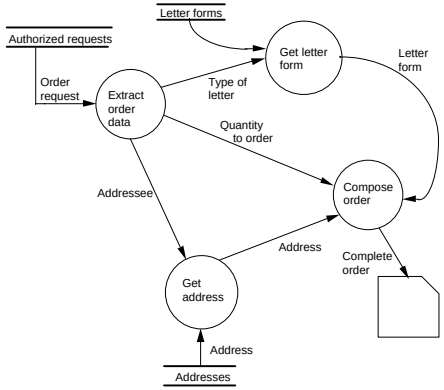


- Possono descrivere:
  - i business process dell'ambiente in cui il sistema andrà inserito
  - Il sistema stesso
- Permettono diversi livelli di astrazione

Flusso di lavoro



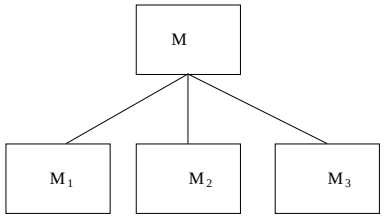
Porzione automatizzata



Dall'analisi al progetto

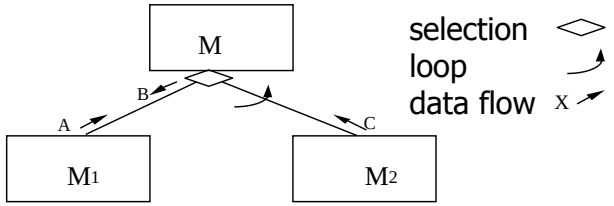
- Notazione: Structured Diagrams (SD)
- SD: Acyclic Directed Graph i cui nodi rappresentano moduli funzionali
- Trasformazione di DFD in SD:
  - minimo accoppiamento
  - massima coesione

SD



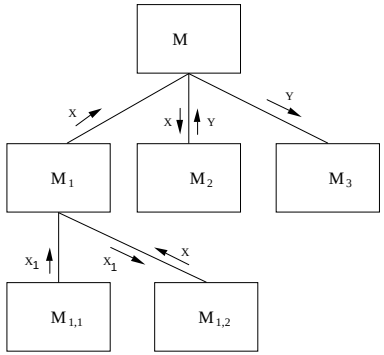
- Ogni modulo rappresenta un'astrazione funzionale.
- M chiama M_1, M_2 ed M_3.

SD decorati



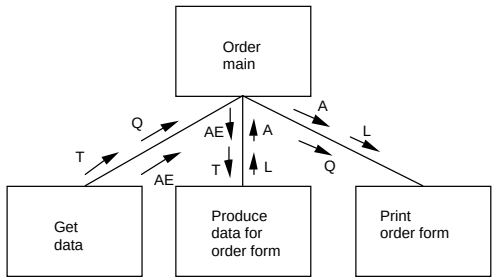
- Parametri:
  - M passa B a M_1 e riceve A
  - M riceve C da M_2
- Controllo:
  - M chiama M_1 o ripetutamente M_2

Esempio



- Funzionalità:
  - M_1 input
  - M_2 trasformazione
  - M_3 output
- A più basso livello:
  - A $M_{1,1}$ input di basso livello
  - A $M_{1,2}$ trasformazione

Esempio di trasformazione



- T = tipo di lettera
- Q = quantità
- AE = indirizzi
- L = modello di lettera
- A = indirizzo

Metodologia di Jackson

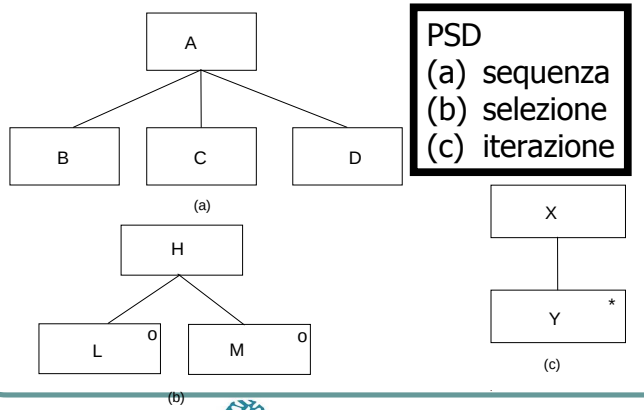
- Jackson System Development (JSD)
- Approcci descrittivi basati su
  - Progettazione orientata agli oggetti
  - Scomposizione funzionale
- Tre fasi:
  - fase di modellazione
  - fase di rete
  - fase di implementazione

Fase di modellazione

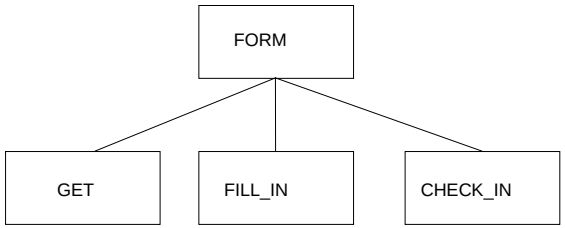
- Dominio applicativo modellato in termini di
  - *entità*: oggetti del mondo reale
  - *azioni* (o *eventi*) che possono influenzare le entità
- Per ogni entità si definiscono le azioni che possono accadere. Esempio:
- Entità FORM, azioni:
  - GET: ottenimento di un modulo
  - FILL_IN: compilazione del modulo
  - CHECK_IN: consegna del modulo compilato

Process Structure Diagram

- Forniscono una vista di processo di un'entità



Esempio

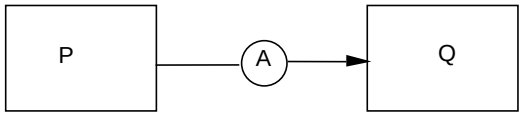


- Il form deve essere, nell'ordine:
  1. ottenuto
  2. compilato
  3. consegnato

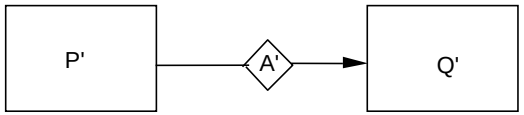
Fase di rete

- Ogni entità è modellata come processo
- Il sistema è rappresentato come rete interconnessa di processi che comunicano fra loro (*System Specification Network*, SSN)
- Forme di comunicazione:
  - stream di dati (coda FIFO di messaggi)
  - vettore di stato (un processo può vedere i dati, cioè lo stato, di un altro)

Fase di rete (SSNs)



(a) Q riceve messaggi da P



(b) Q' esamina lo stato di P'

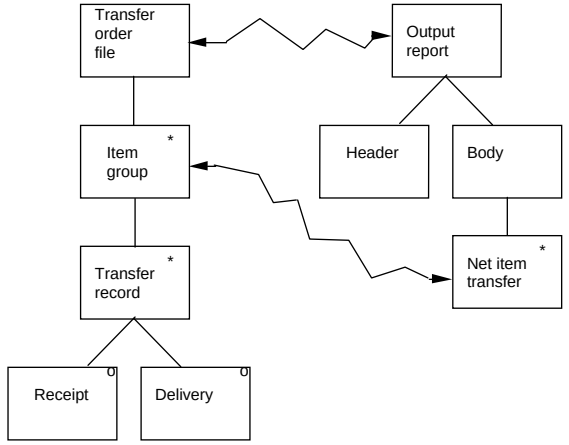
Fase di implementazione

- Trasforma la rete di processi in un'implementazione.
- Se necessario, si trasforma in un sistema sequenziale (*inversione di processo*).
- Esempio (caso (a) della slide precedente):
  - P produce il dato, passa il controllo a Q e poi lo riprende
  - Quando Q ha bisogno di un dato, passa il controllo a P e, quando il dato è prodotto, lo riprende

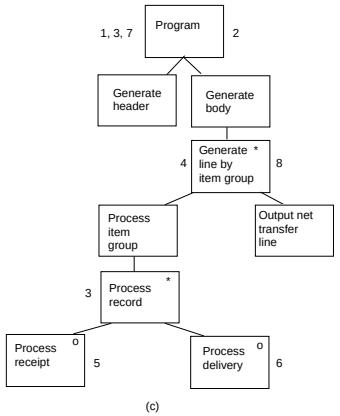
Jackson Structured Programming (JSP)

- Stabilisce la struttura di un programma data la struttura del suo input e del suo output
- Data la struttura di input e output:
  - si individuano le corrispondenze
  - si definisce la struttura del programma
  - si elencano le operazioni da eseguire e le si posiziona nella struttura del programma
- La notazione può essere usata per chiarire ulteriormente i PSD
- Adatto allo sviluppo di programmi tradizionali per la gestione di file (ad es. Cobol)

Esempio: struttura di input e output



Esempio: struttura del programma



1. open files
2. close files
3. read (item_id, movement)
4. net_item_movement := 0
5. net_item_movement := net_item_movement + movement
6. net_item_movement := net_item_movement - movement
7. write (header)
8. write (net_item_movement_line)

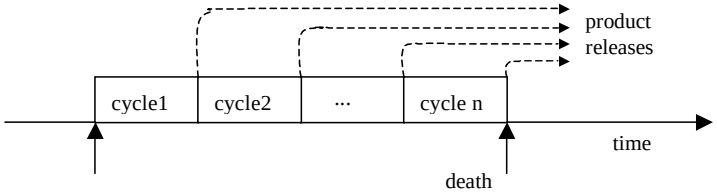
Unified software development Process (UP)

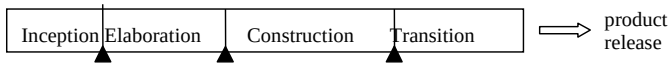
- Introdotto da Ericsson negli anni Sessanta
- Standard di fatto
- Basato su UML
- Processo iterativo e incrementale

UP

- Scompone un processo di grandi dimensioni in una sequenza di *iterazioni controllate* (mini progetti)
- Una nuova iterazione deve portare a:
  - implementazione di nuovi use case
  - la gestione di un rischio critico
- Il ciclo di vita UP è una sequenza di cicli, ognuno dei quali porta a una *versione rilasciabile* del prodotto.

Cicli e fasi



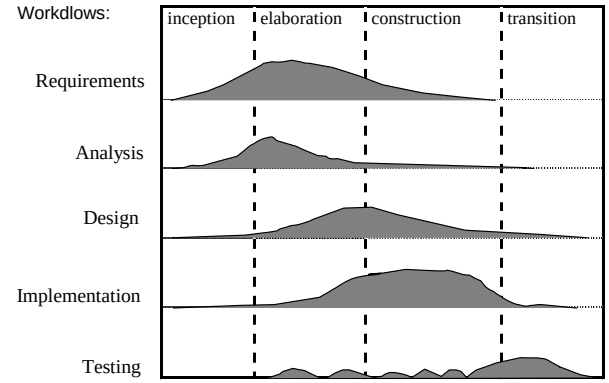


milestone ▲

Fasi

- Ogni ciclo è formato da quattro fasi, ognuna delle quali termina con una milestone:
  - Concezione: simile a studio di fattibilità (utenti potenziali, architettura preliminare, piano di sviluppo)
  - Elaborazione: definizione di use case e progettazione architetturale (*baseline*)
  - Costruzione: il sistema viene sviluppato e testato secondo la baseline
  - Transizione: beta test

Suddivisione delle attività nei cicli in UP



Differenza tra requirements e analysis: analysis raffina e struttura i requisiti per migliorare la comprensione, la manutenibilità e gli sviluppi futuri

Caratteristiche di UP

- Flessibile, ma strutturato
- Incrementale
- Consente convalida continua
- Guidato da use case: usati anche come mezzo di comunicazione con utenti e stakeholder
- Centrato sulle architetture: l'architettura (risultato della fase di elaborazione) è l'artefatto principale.

Metodi agili

- Estremizzazione delle caratteristiche dei modelli evolutivi
- Criticano il sovraccarico dovuto a specifica rigorosa, progettazione, documentazione
- Basati sulla codifica, più che sul progetto
- Hanno lo scopo di rendere rapide
  - la consegna di un prodotto funzionante
  - la reazione a modifiche dei requisiti

Principi dei metodi agili

- Coinvolgimento del cliente:
  - Priorità dei requisiti
  - Convalida delle versioni incrementali
- Consegna incrementale
- Persone, anziché processi: libera espressione delle capacità individuali
- Accettazione del cambiamento:
  - i cambiamenti dei requisiti sono parte del processo e non un inconveniente
  - il prodotto e il processo sono studiati in modo da reagire prontamente
- Semplicità: del prodotto e del processo

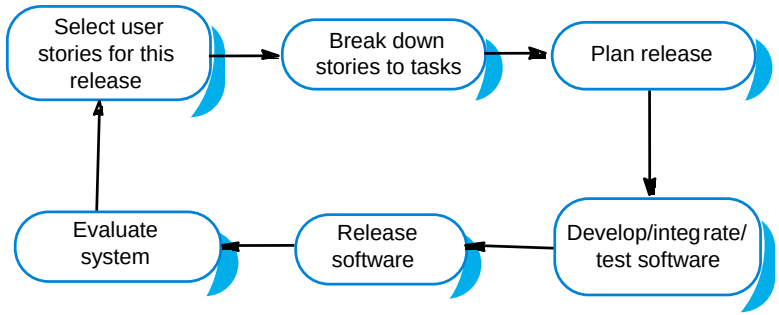
Valutazione

- Problemi:
  - Difficile mantenere coinvolti i clienti
  - Richiede un coinvolgimento intenso degli sviluppatori
  - Le priorità nei requisiti possono non essere facili da determinare (molti stakeholder)
  - La semplicità richiede ulteriore lavoro
  - Difficile integrare i requisiti nel contratto
- Probabilmente adatti per piccoli progetti

Extreme Programming (XP)

- Il metodo agile più diffuso
- Estremizza lo sviluppo iterativo:
  - Il sistema può essere assemblato anche più di una volta al giorno
  - Le versioni incrementali sono consegnate ogni due settimane
  - Tutte le versioni (anche non rilasciate) devono superare tutti i test

Ciclo di rilascio in XP



Pratiche di XP

- Pianificazione incrementale:
  - i requisiti sono registrati (story cards)
  - si includono o meno in un rilascio secondo il tempo disponibile e la priorità
- Rilasci minimi:
  - si parte dalla consegna di un insieme minimo di funzionalità utili
  - funzionalità ulteriori si aggiungono nei rilasci successivi
- Progetto semplice:
  - a ogni momento, il progetto è il più semplice che permetta di implementare i requisiti che si è deciso di soddisfare
  - si complica solo se necessario per soddisfare ulteriori requisiti

Pratiche di XP

- Sviluppo “test first”:
  - Strumenti per unit testing automatizzato
  - I test di un modulo sono progettati *prima* dell’implementazione
- Refactoring:
  - appena si trova un modo per rendere il codice più semplice mantenendone la funzionalità, esso va riscritto
- Programmazione in coppia:
  - Gli sviluppatori lavorano in coppia, supportandosi e controllandosi a vicenda
- Proprietà collettiva del codice:
  - Tutte le coppie lavorano su tutte le parti del codice, in modo che la conoscenza sia condivisa
  - Il codice appartiene a tutti; chiunque può fare modifiche ovunque

Pratiche di XP

- Integrazione continua:
  - Appena un modulo è pronto, va integrato nel sistema
  - Per ogni assemblaggio, vanno ripetuti tutti i test
- Ritmo sostenibile:
  - Ritmi eccessivi di lavoro vanno evitati, in quanto il risultato netto è la riduzione della qualità del prodotto e della produttività a medio termine
- Cliente on-site:
  - Il cliente o un suo rappresentante dovrebbe essere disponibile a tempo pieno
  - Fa parte della squadra di sviluppo, con lo scopo di indicare i requisiti e le priorità fra requisiti

Principi dei metodi agili e XP

- Sviluppo incrementale: rilasci frequenti
- Coinvolgimento del cliente: cliente on-site
- Persone anziché processi: programmazione a coppie, proprietà collettiva e ritmi sostenibili
- Cambiamento: rilasci frequenti, implementazione di nuovi requisiti
- Semplicità: refactoring continuo

Requisiti

- Espressi come scenari o user stories
- Sono registrati su schede (story card) e il team di sviluppo li scompone in task di implementazione (funzionalità), che determinano le stime di costi e tempi
- Il cliente sceglie le stories da implementare in un rilascio secondo le priorità e le stime

Esempio: story card

Downloading and printing an article
First, you select the article that you want from a displayed list. You then have to tell the system how you will pay for it - this can either be through a subscription, through a company account or by credit card. After this, you get a copyright form from the system to fill in and, when you have submitted this, the article you want is downloaded onto your computer You then choose a printer and a copy of the article is printed. You tell the system if printing has been successful. If the article is a print-only article, you can't keep the PDF version so it is automatically deleted from your computer

Anticipazione del cambiamento in XP

- Il principio di anticipazione del cambiamento, ampiamente accettato, prevede di investire nella previsione dei cambiamenti possibili, per ridurre i costi collegati.
- In XP, questo investimento è considerato inutile, in quanto non è possibile prevedere realisticamente i cambiamenti.
- XP prevede invece di mantenere il codice semplice tramite refactoring, per rendere i cambiamenti meno costosi, quando se ne manifesterà la necessità

Test in XP

- Sviluppo “test first”.
- Test progettati a partire dagli scenari (black box) prima dell’implementazione delle funzionalità.
- Ogni task (implementazione di una funzionalità prevista nello scenario) verrà testato per mezzo di uno o più casi di test
- Lo progettazione e la convalida dei test prevedono il coinvolgimento del cliente
- Tutti i test sono eseguiti tramite procedure automatiche a ogni assemblaggio del sistema

Task per il download di documenti

Task 1: Implement principal workflow

Task 2: Implement article catalog and selection

Task 3: Implement payment collection

Payment may be made in 3 different ways. The user selects which way they wish to pay. If the user has a library subscription, then they can input the subscriber key which should be checked by the system. Alternatively, they can input an organisational account number. If this is valid, a debit of the cost of the article is posted to this account. Finally, they may input a 16 digit credit card number and expiry date. This should be checked for validity and, if valid a debit is posted to that credit card account.

Descrizione di un caso di test

Test 4: Test cr edit card validity

Input:
A string representing the credit card number and two integers representing the month and year when the card expires
Tests:
Check that all bytes in the string are digits
Check that the month lies between 1 and 12 and the year is greater than or equal to the current year .
Using the first 4 digits of the credit card number , check that the card issuer is valid by looking up the card issuer table. Check credit card validity by submitting the card number and expiry date information to the card issuer
Output:
OK or error message indicating that the card is invalid

Programmazione a coppie

- Favorisce la proprietà comune del codice e la diffusione di conoscenza nel team
- Serve come processo di revisione informale
- Incoraggia il refactoring, di cui beneficia l'intero team
- Le misure effettuate indicano che la produttività di una coppia di sviluppatori è simile a quella di due sviluppatori che lavorano separatamente.

Microsoft: Synchronize & Stabilize

- Studio di Cusumano e Selby (1995, 1997).
- Scomposizione dei progetti in piccole squadre che
  - lavorano in parallelo
  - hanno grande autonomia
- Obiettivo: velocità di sviluppo

Fase di pianificazione

- Visione
  - Definisce gli obiettivi generali del prodotto
  - Stabilisce priorità fra le caratteristiche
- Specifica
  - Requisiti, architettura globale, interdipendenze fra componenti
  - Flessibile (30% cambia)
- Tabella di marcia

Fase di sviluppo

- Squadra composta da:
  - sviluppatori
  - tester (test continuo)
- Sincronizzazione giornaliera
  - *Build* giornaliero spedito al test
- Stabilizzazione
  - *Milestone*: date a cui il prodotto deve arrivare con determinate qualità

Open source

- Disponibilità del codice sorgente, per consentire modifiche
- Le licenze (es. GPL) regolano:
  - uso
  - copia
  - modifica
  - distribuzione

  generalmente in modo poco restrittivo, ma alcune vincolano le versioni successive a mantenere la stessa licenza.

Modello di sviluppo

- Processo distribuito
  - Gruppo ristretto di sviluppatori
  - Apertura a un grande numero di contributors
  - Disponibilità del codice sorgente in ogni fase
- Coordinazione attraverso Internet
  - Mailing list
  - Repository condivise (CVS, SVN)

Confronto con altri modelli di business

- La fonte di redditività non è lo sviluppo, ma i servizi, la formazione, la personalizzazione
- Prodotti di buona qualità se con ampia base di utenti
- Utenti: beta tester, spesso non segnala solo i problemi ma li risolve

Confronto con altri modelli di business

- Linus Torvalds: la fase di debugging può procedere in parallelo (la coordinazione richiesta è ridotta)
- Critiche:
  - perdita di integrità concettuale
  - difficile la coordinazione necessaria per prodotti complessi e in continua evoluzione